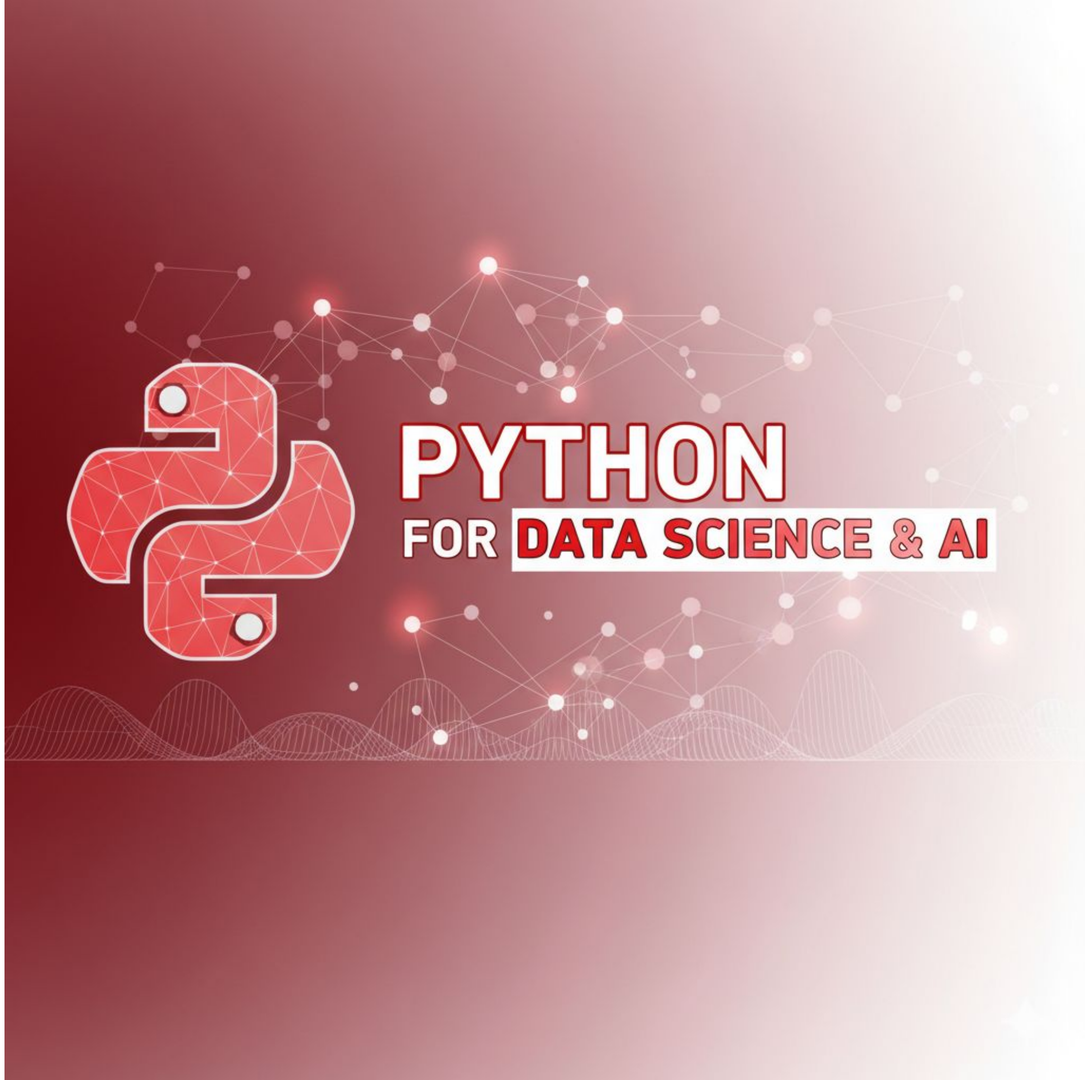
Python for Data Science & AI

Dasun Athukoralage
web: www.dasuna.me
email: dasun@nirvanaclouds.com

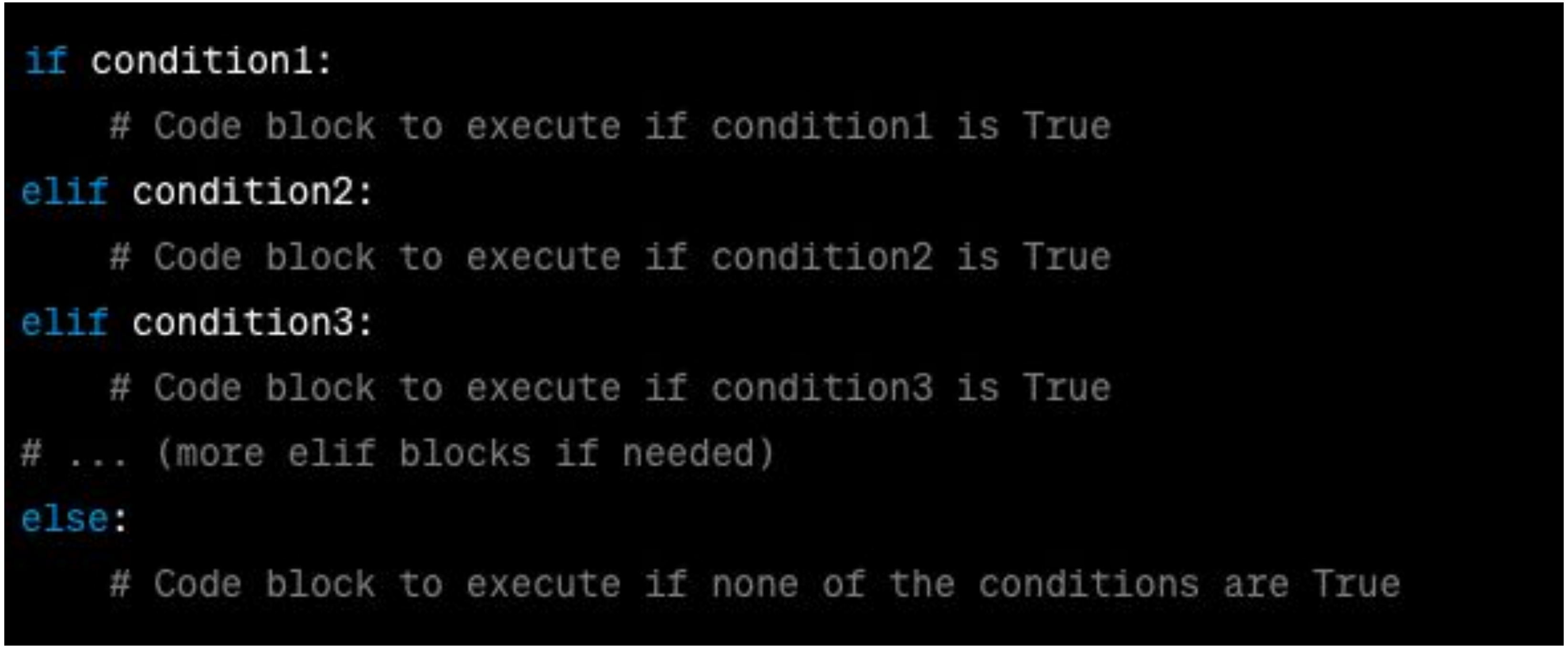


Python if-elif-else Statement

This allows you to handle multiple conditions sequentially.

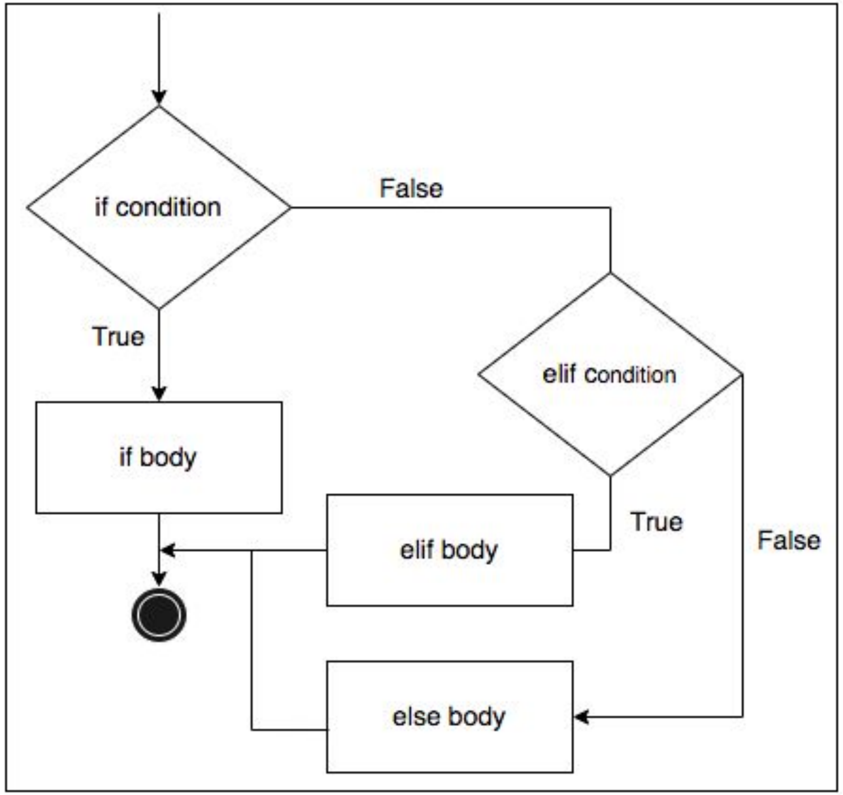
It stands for "**if-else if-else**" and is used when you have a series of conditions to check, and you want to execute different blocks of code based on which condition is satisfied. This construct provides a way to create a chain of conditions and associated actions.

Python if-elif-else Statement



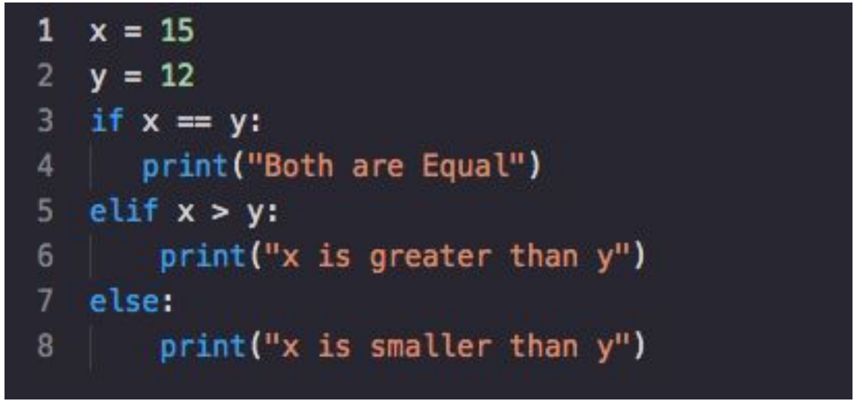
```
if condition1:
    # Code block to execute if condition1 is True
elif condition2:
    # Code block to execute if condition2 is True
elif condition3:
    # Code block to execute if condition3 is True
# ... (more elif blocks if needed)
else:
    # Code block to execute if none of the conditions are True
```

Python if-elif-else Statement



Python if-elif-else Statement

In the below code provided, the initial if statement checks whether 15 is equal to 12. If not, the elif condition will verify that 15 is greater than 12. If this condition is true, the code block inside elif will be executed. Otherwise, the code in the else block will be executed if none of the preceding conditions are met.



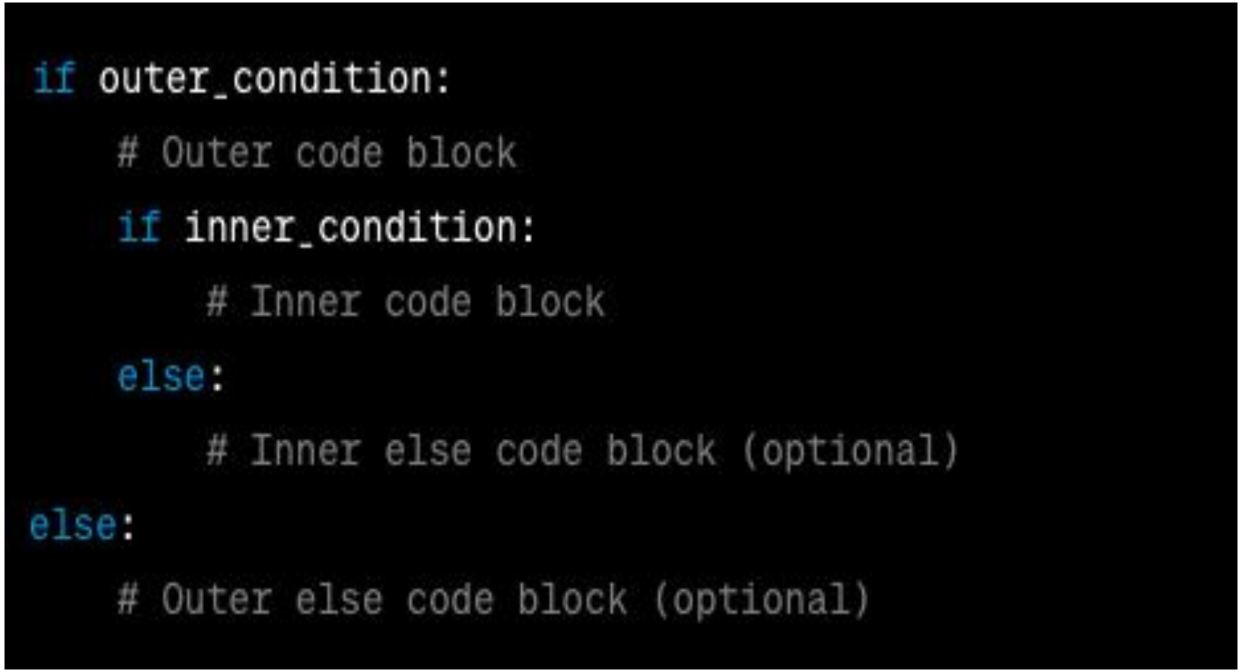
```python
x = 15
y = 12
if x == y:
    print("Both are Equal")
elif x > y:
    print("x is greater than y")
else:
    print("x is smaller than y")
```

Nested if Statement

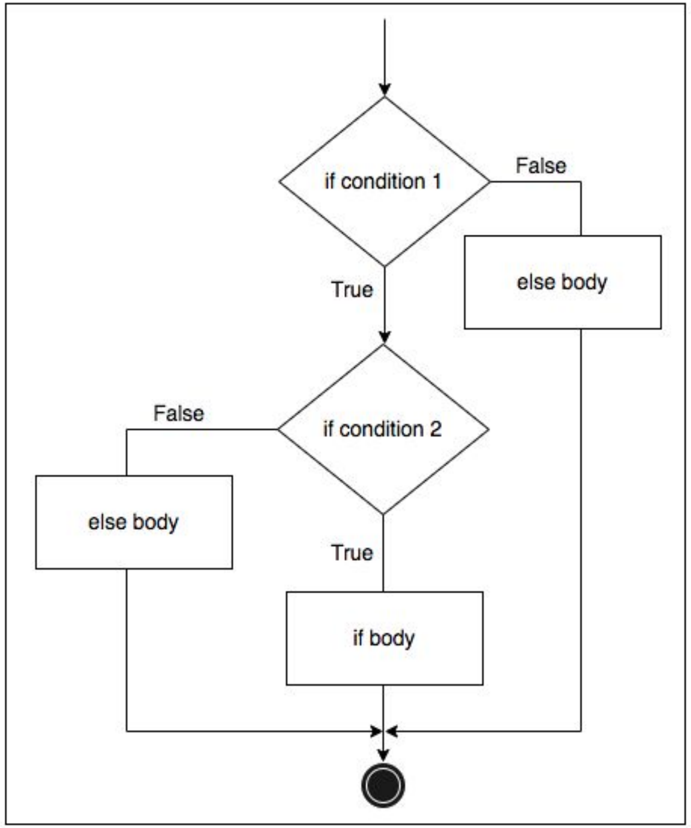
A nested if statement is a control flow construct in programming, specifically in Python, where **an if statement is placed inside another if statement**.

The inner if statement is evaluated only if the outer if statement's condition is true.

Nested if Statement

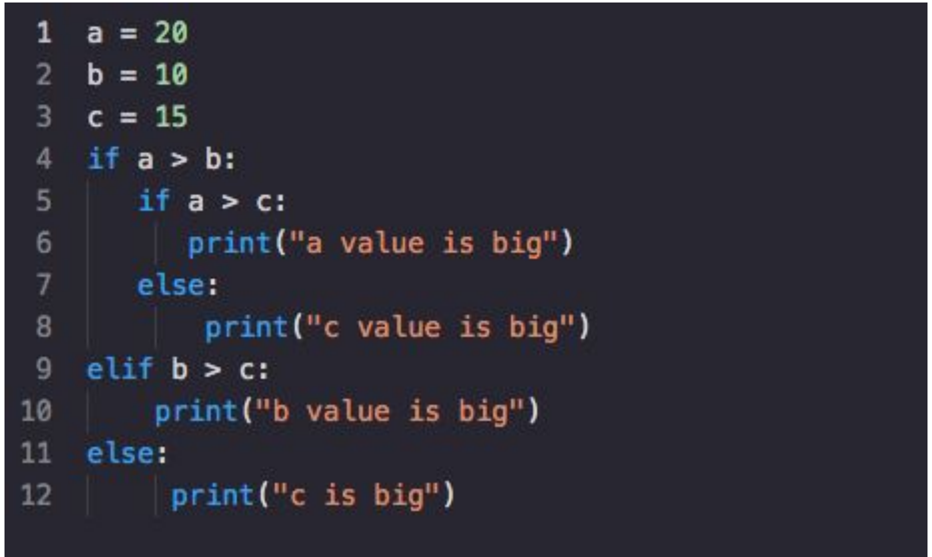

```
if outer_condition:
    # Outer code block
    if inner_condition:
        # Inner code block
    else:
        # Inner else code block (optional)
else:
    # Outer else code block (optional)
```

Nested if Statement



Nested if Statement

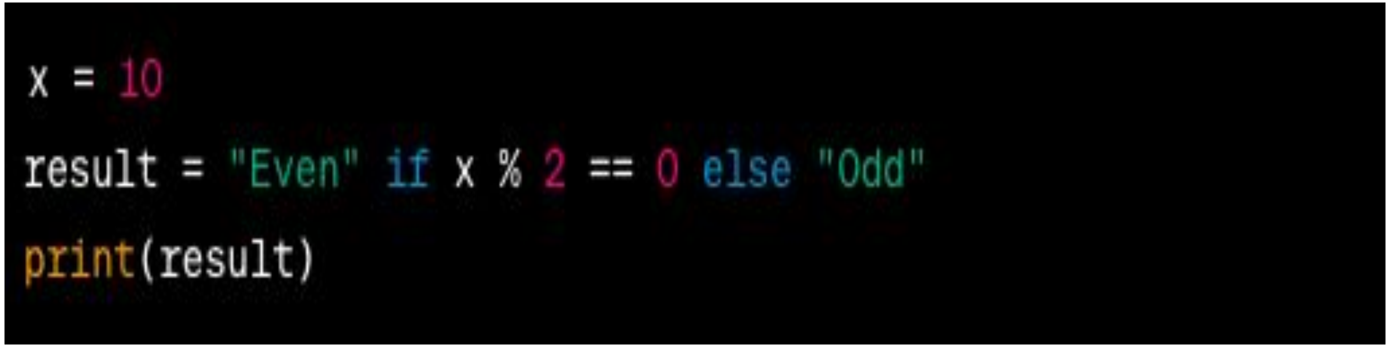
In the following code example, we can see first if condition checks a is greater than b. If yes, then we've another if condition that checks a is also greater than c. If yes, then if body will be executed.



```python
a = 20
b = 10
c = 15
if a > b:
    if a > c:
        print("a value is big")
    else:
        print("c value is big")
elif b > c:
    print("b value is big")
else:
    print("c is big")
```

Ternary Operator in Python

It simply allows testing a condition in a **single line** replacing the multiline if-else making the code compact.



```
x = 10
result = "Even" if x % 2 == 0 else "Odd"
print(result)
```

Structural Pattern Matching

From **version 3.10 upwards**, Python has implemented a switch case feature called “**structural pattern matching**”. You can implement this feature with the **match** and **case** keywords.

This is even more powerful than simple switch-case: you can **match patterns, structures, and types**.

Structural Pattern Matching

[Basic Syntax]

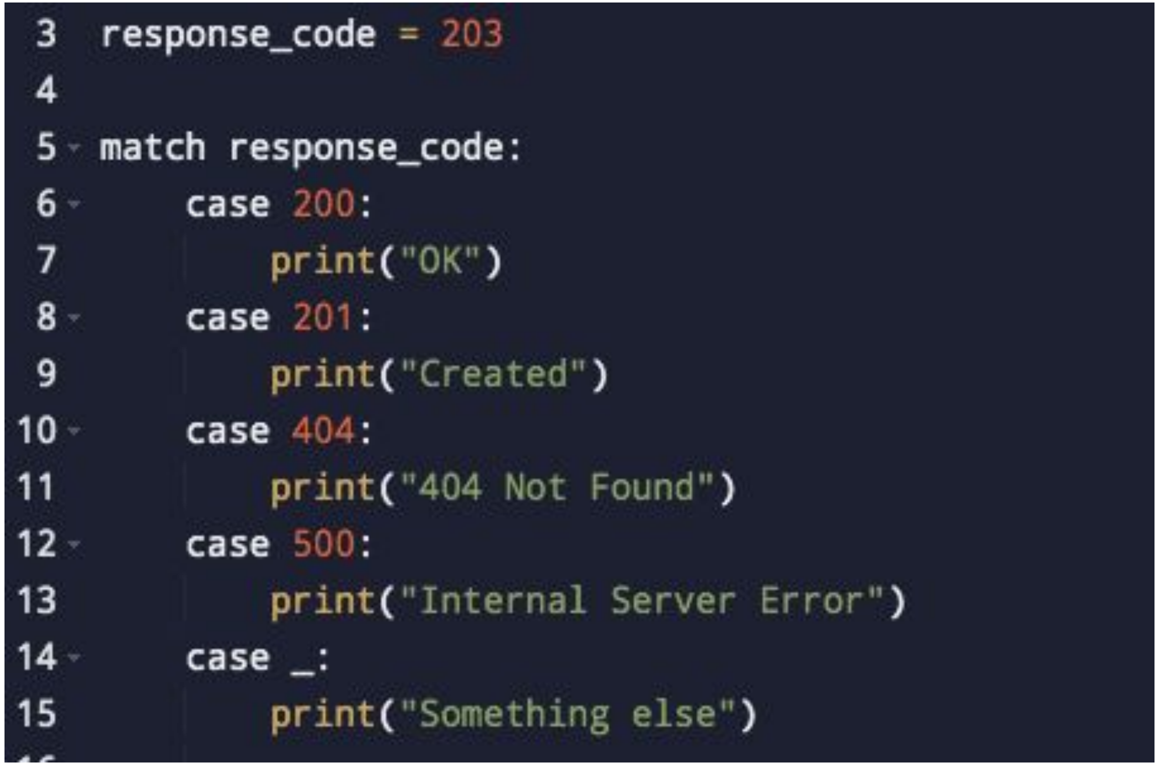
```
match variable:
    case pattern1:
        # code block
    case pattern2:
        # code block
    case _:
        # default block
```

Structural Pattern Matching

If the value of ‘variable’ matches one of the patterns, the corresponding block of code is executed.

If the value of ‘variable’ does not match any of the patterns, **the default (i.e. case _:)** block of code is executed.

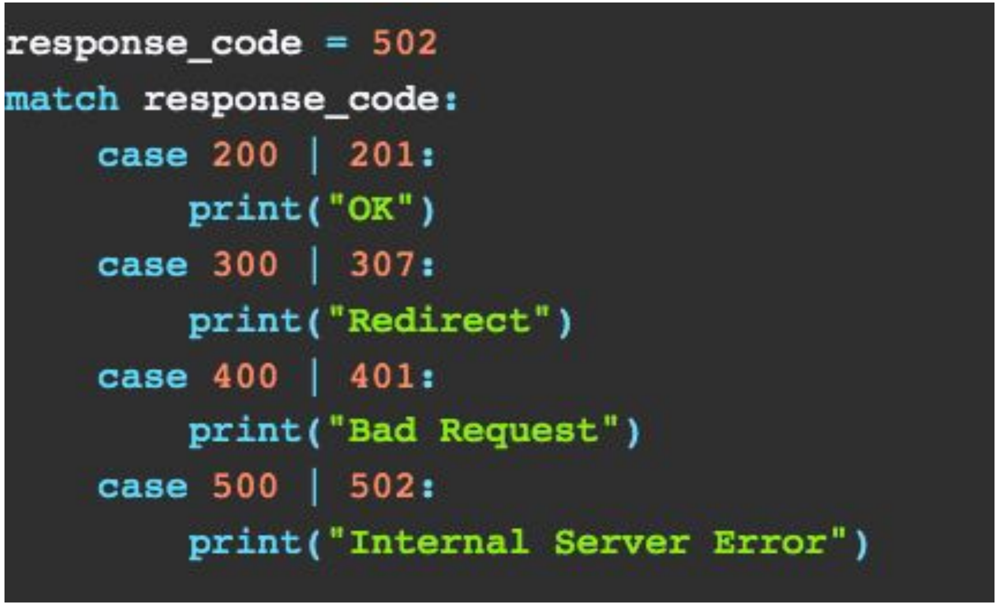
Structural Pattern Matching



```python
response_code = 203

match response_code:
    case 200:
        print("OK")
    case 201:
        print("Created")
    case 404:
        print("404 Not Found")
    case 500:
        print("Internal Server Error")
    case _:
        print("Something else")
```

Structural Pattern Matching

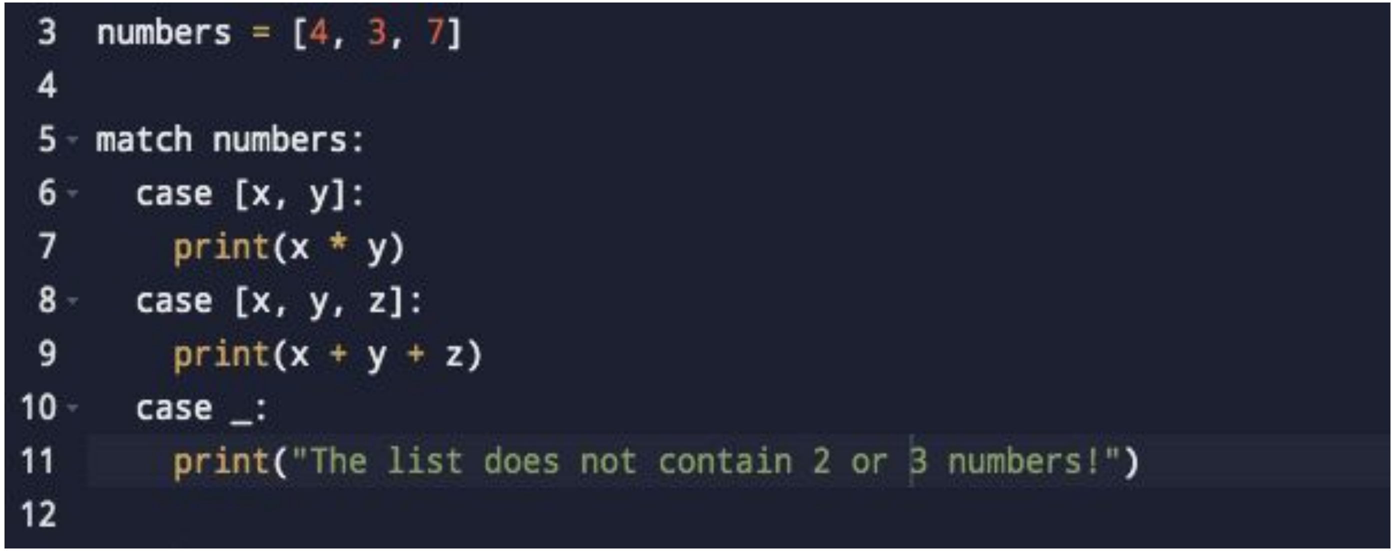
Matching with the or pattern



```
response_code = 502
match response_code:
    case 200 | 201:
        print("OK")
    case 300 | 307:
        print("Redirect")
    case 400 | 401:
        print("Bad Request")
    case 500 | 502:
        print("Internal Server Error")
```

Structural Pattern Matching

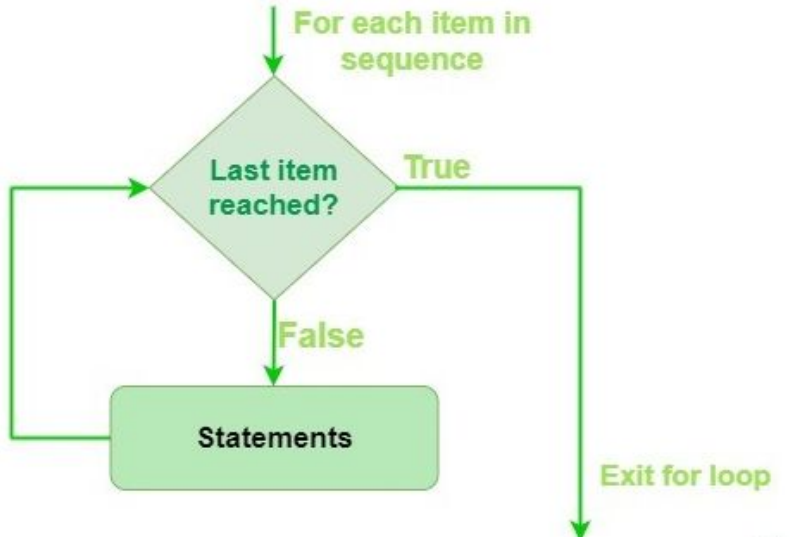
[Matching by the length of an Iterable]



```
numbers = [4, 3, 7]

match numbers:
  case [x, y]:
    print(x * y)
  case [x, y, z]:
    print(x + y + z)
  case _:
    print("The list does not contain 2 or 3 numbers!")
```

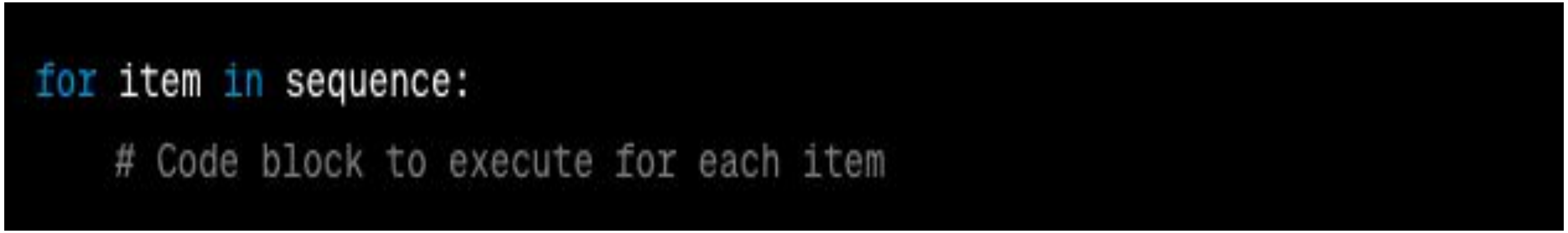
Loops

Loops are fundamental programming constructs that allow you to **repeatedly execute a block of code as long as a certain condition is met.**



for Loop

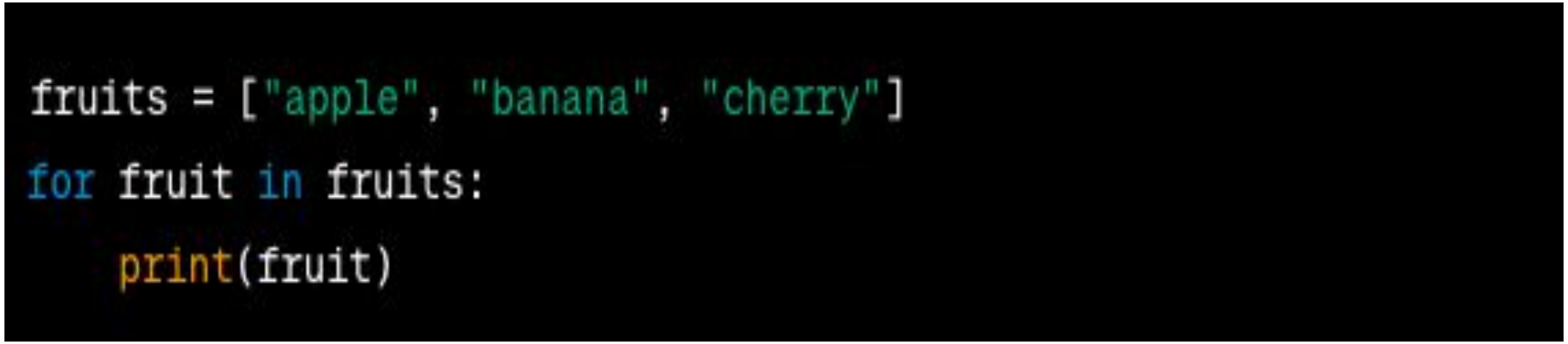
The for loop is used to **iterate over a sequence (such as a list, tuple, string, or range)** and execute a block of code for each item in the sequence.



```
for item in sequence:
    # Code block to execute for each item
```

for Loop

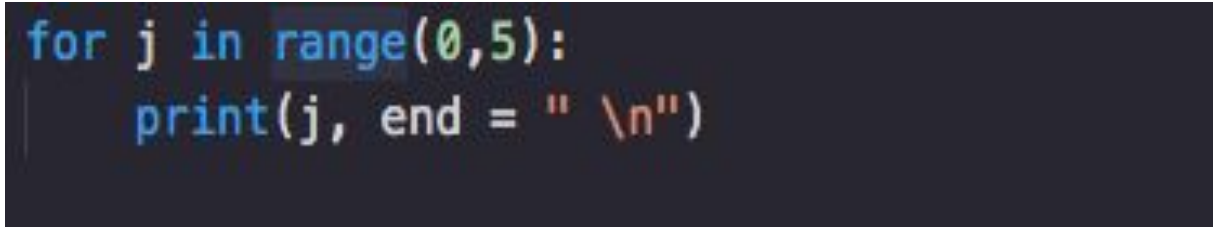
Example 1:



```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
```

for Loop

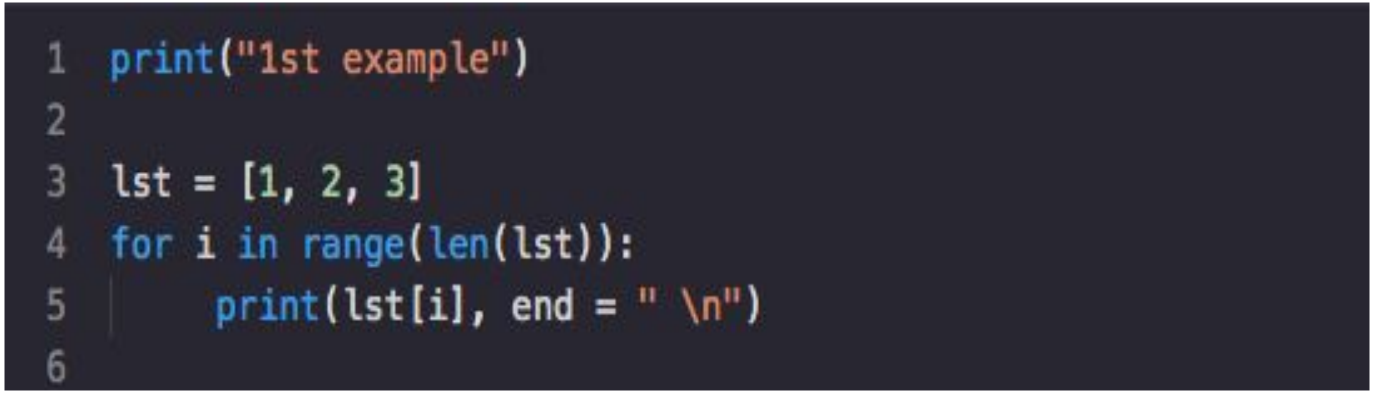
Example 2:



```
for j in range(0,5):
    print(j, end = " \n")
```

for Loop

Example 3:



```
print("1st example")

lst = [1, 2, 3]
for i in range(len(lst)):
    print(lst[i], end = " \n")

```

for Loop

List Comprehension:

list comprehension is **a concise way to generate lists** by iterating over an iterable and applying an expression to each element.

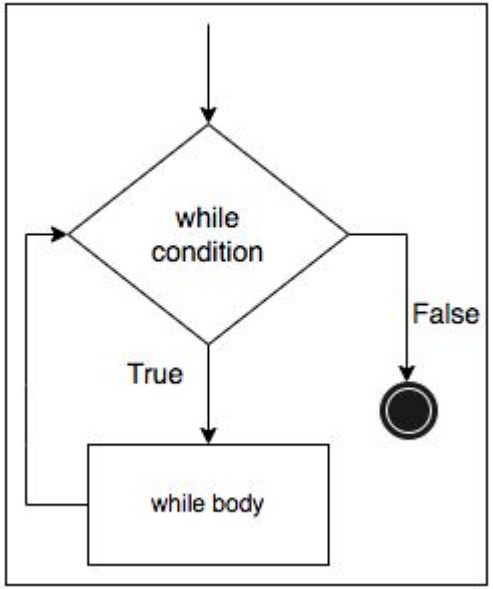
```
[expression for item in iterable if condition]
```

Here:

- **expression** is the value that gets added to the list.
- **item** is the current element from the iterable.
- **if condition** is optional and filters elements.

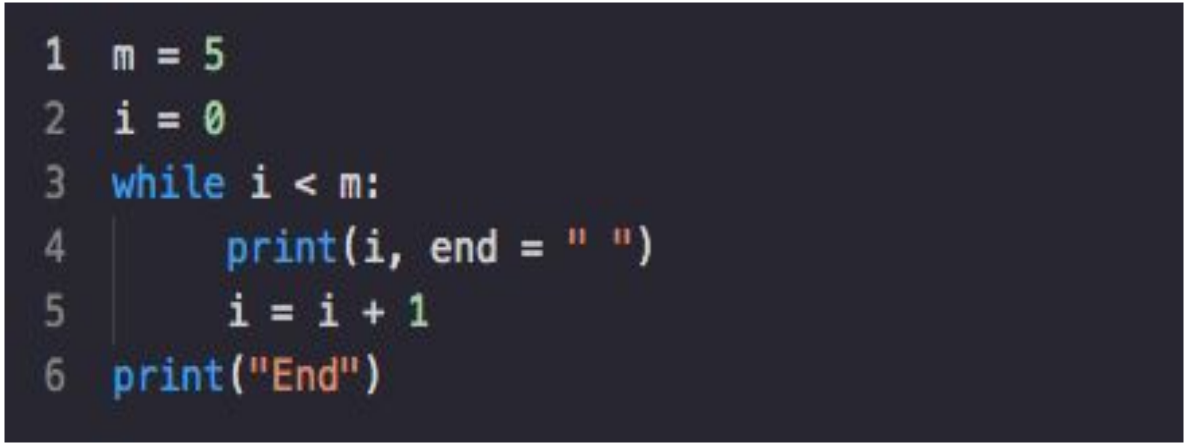
while Loop

The while loop repeatedly executes a block of code as long as a certain condition is true.



while Loop

The following code iterates from 0 to 4 and prints each value using a while loop. It prints End to signify the end of the program after the loop is completed.



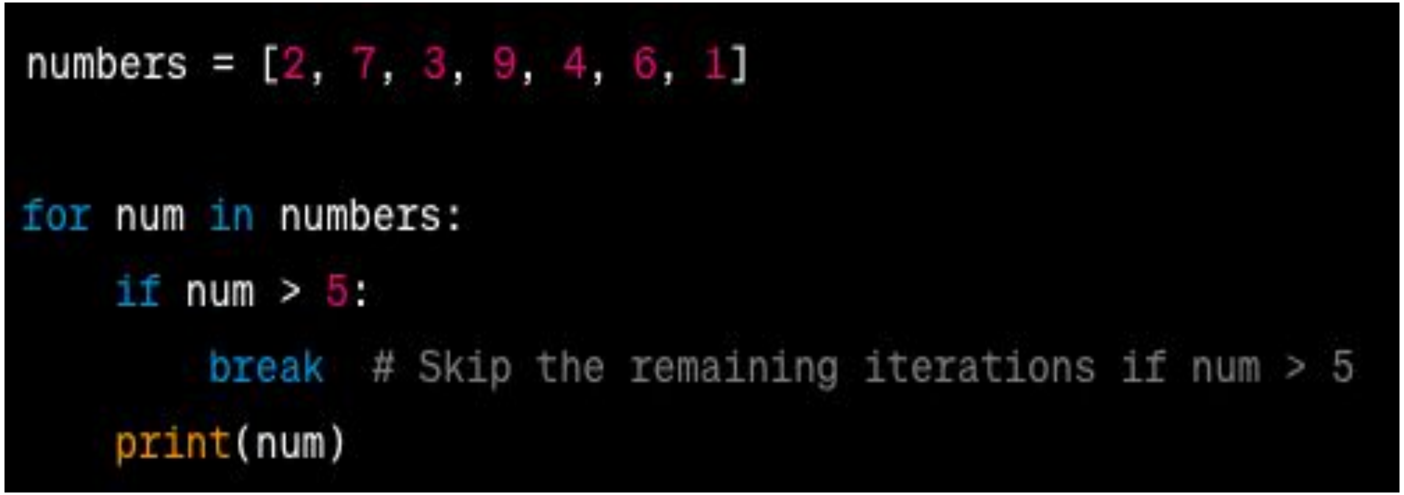
```
m = 5
i = 0
while i < m:
        print(i, end = " ")
        i = i + 1
print("End")
```

Exiting from a loop using ‘break’

The break statement is used to **immediately exit the loop when a certain condition is met**.

In this example, we'll use a for loop to iterate through a list of numbers. We want to skip iterations for numbers greater than 5.

Exiting from a loop using ‘break’



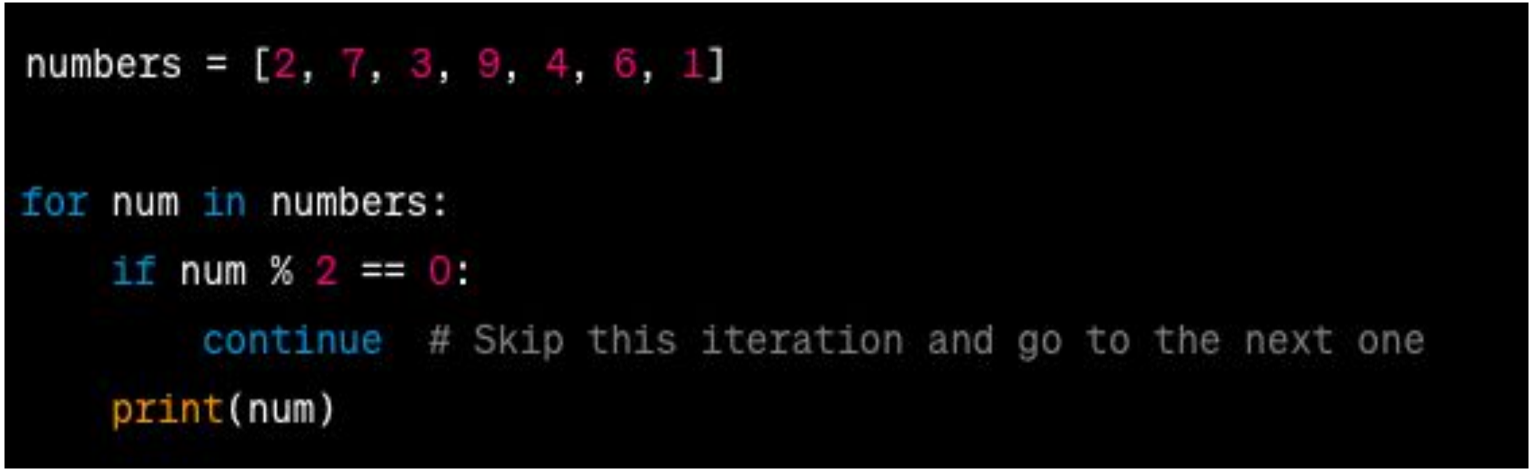
```
numbers = [2, 7, 3, 9, 4, 6, 1]

for num in numbers:
    if num > 5:
        break  # Skip the remaining iterations if num > 5
    print(num)
```

Exiting from a loop using ‘continue’

The continue statement **skips the current iteration and moves on to the next one**.

In this example, we'll use a for loop to iterate through a list of numbers. We want to skip printing even numbers.

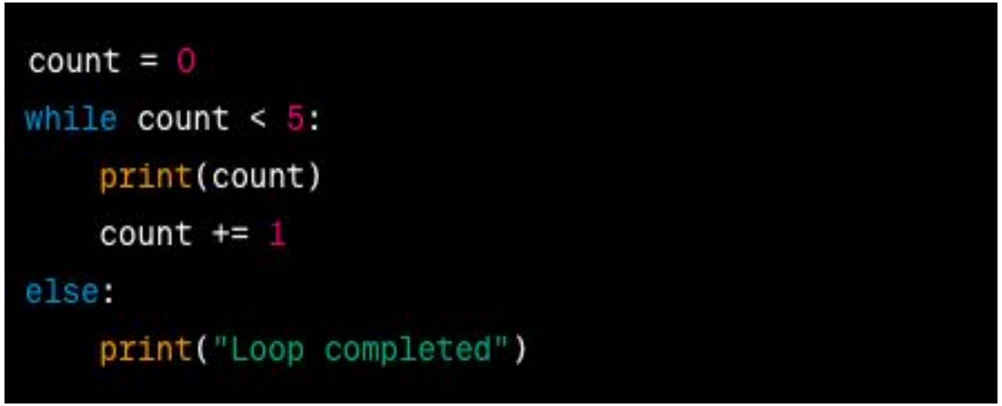
Exiting from a loop using ‘continue’



```
numbers = [2, 7, 3, 9, 4, 6, 1]

for num in numbers:
    if num % 2 == 0:
        continue  # Skip this iteration and go to the next one
    print(num)
```

while Loop with ‘else’

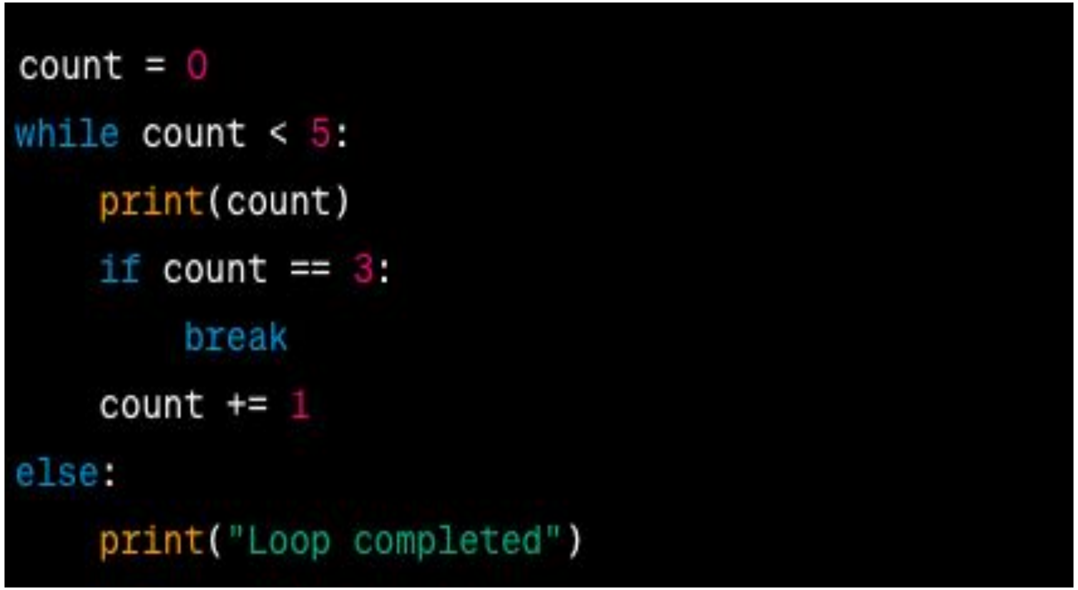
This construct allows you to specify a block of code that should be executed once the while loop has completed its iterations, provided that the loop was not exited prematurely using a break statement.



```
count = 0
while count < 5:
    print(count)
    count += 1
else:
    print("Loop completed")
```

What’s the output?

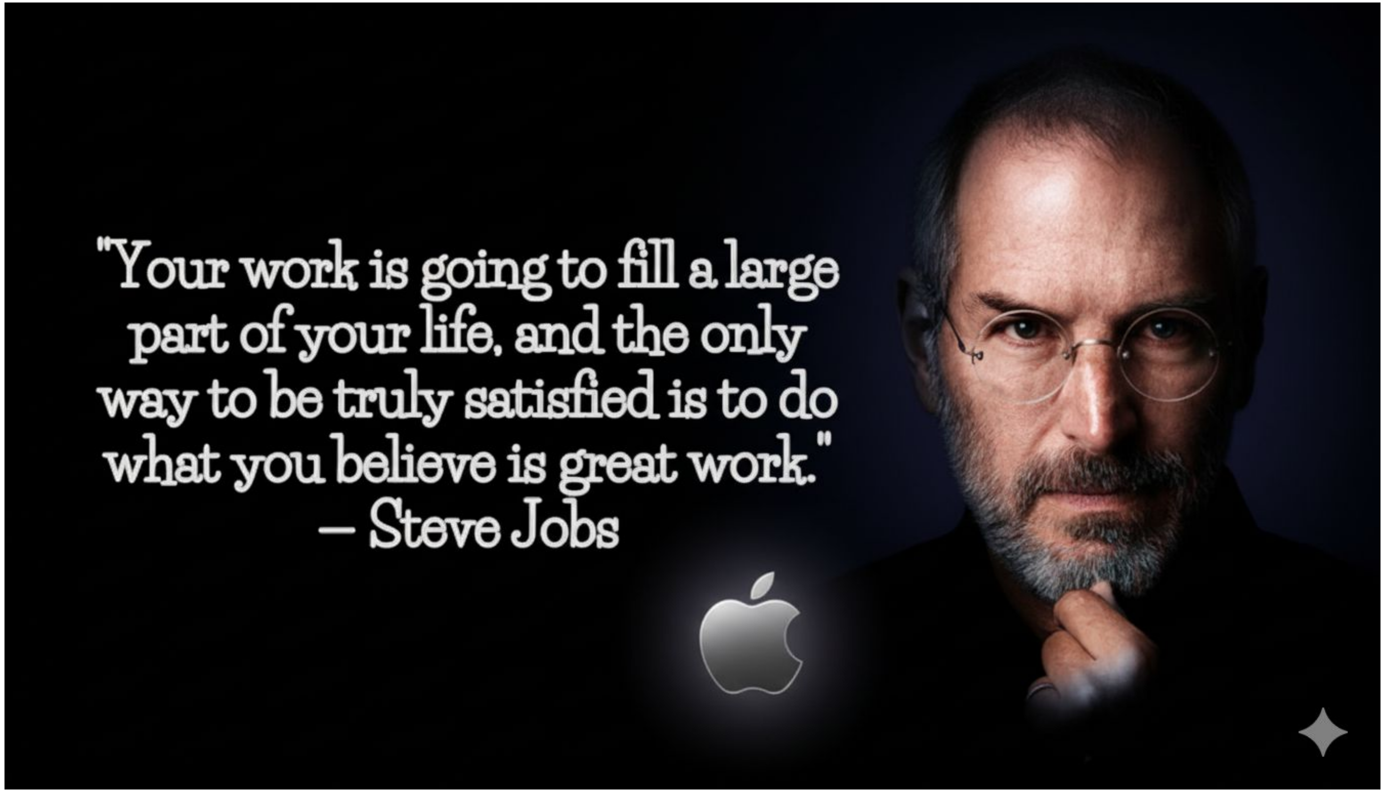
while Loop with ‘else’



```
count = 0
while count < 5:
    print(count)
    if count == 3:
        break
    count += 1
else:
    print("Loop completed")
```

What’s the output?

Thank You...!


"Your work is going to fill a large part of your life, and the only way to be truly satisfied is to do what you believe is great work."
– Steve Jobs